occlusion_reflection_detec.h File Reference

individuazione dei riflessi e occlussioni More...

```
#include <opencv2/imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <math.h>
#include <vector>
#include "polynomial_regression.h"
```

Go to the source code of this file.

Macros

#define	**NUM_RAYS**	46
#define	**KSIZE**	3

Functions

void	**drawRays** (Mat *norm_img) sull'iride normalizzata di input vengono disegnati NUM_RAYS (46, in base all'header) partendo dalla posizione centrale superiore, ovvero partendo da (x,y) = (width/2,0). More...
void	**initKernels** () viene inizializzato il kernel da applicare tramite convoluzione all'immagine normalizzata sul quale sono stati precedentemente disegnati i raggi per l'individuazione della palpebra superiore. N.B.: è qui stato utilizzato il kernel differenziale di prewitt 3x3, ma è possibile utilizzare anche sobel o scharr scharr e sobel possono essere creati direttamente con la funzione OpenCV "getDerivKernels". Vedi la documentazione per maggiori informazioni a riguardo.
void	**initRayPos** (Mat *normImg, Mat *rayPos) viene segnata la posizione di tutti i pixel facenti parte dei 46 raggi precedentemente disegnati sull'immagine normalizzata More...
double	**pixelConvolution** (Mat *normImg, int x, int y, int ray) convoluzione con il kernel differenziale del pixel (x,y) del raggio numero ray disegnato precedentemente sull'immagine normalizzata normImg More...
Mat	**upperEyelidDetection** (Mat *normImg, string path) viene localizzata e individuata la palpebra superiore. Viene fatto disegnando sullo spettro rosso dell'immagine normalizzata dei raggi al quale verrà applicato, tramite convoluzione, un kernel differenziale (es: prewitt). I valori risultanti da tale convoluzione vengono valutati e per ogni raggio viene preso il valore massimo. Viene effettuata una scremetura di possibili outliers (questa fase sarebbe da migliorare) e viene applicata ai valori restanti la regressione polinomiale (terzo ordine). Una volta ottenuto il polinomio di terzo ordine, vengono calcolati i valori per la curva della palpebra superiore. Se non è possibile disegnare tale curva, si procede con la funzione ellipseFitting, in cui vengono presi anche i maxima speculari (y*-1) e viene disegnata un ellisse. Alla fine di tutto viene creata la

	maschera binaria di output e viene swappata per garantire una corretta posizione della palpebra inferiore e superiore per effettuare un merge di tutte le maschere. More...
vector< double >	**localMinima** (vector< double > vec) filtro i valori che rappresentano dei minimi locali, in cui presi tre valori ho vec[i-1] > vec[i] < vec[i+1] More...
void	**removeOutliers** (vector< int > *vec_x, vector< int > *vec_y) elimino gli outliers (pixel i) in cui, presi tre pixel i-1, i, i+1 la posizione del pixel i ha la coordinata x[i-1] > x[i] < x[i+1] oppure x[i-1] < x[i] > x[i+1], ovvero elimino tutti i pixel che hanno un incremento o decremento improvvisi sull'asse delle x. Successivamente elimino anche i pixel in cui y[i-1] > y[i] < y[i+1] oppure y[i-1] < y[i] > y[i+1], ovvero che hanno improvvisi incrementi o decrementi sull'asse delle y More...
Mat	**lowerEyelidDetection** (Mat *normImg) tramite l'utilizzo della mean e standard deviation viene calcolato un valore di threshold. Tale valore viene usato per individuare la palpebra inferiore, operazione che viene effettuata solo se la standard deviation è > della mean deviation /4 More...
Mat	**threshReflectionDetection** (Mat *normImg, int ksize, double c) tramite adaptive thresholding vengono individuati i riflessi all'interno dell'immagine normalizzata (spettro blu) More...

Detailed Description

individuazione dei riflessi e occlussioni

Function Documentation

◆ drawRays()

void drawRays (Mat * norm_img)

sull'iride normalizzata di input vengono disegnati NUM_RAYS (46, in base all'header) partendo dalla posizione centrale superiore, ovvero partendo da (x,y) = (width/2,0).

Parameters

norm_img immagine normalizzata a singolo spettro (spettro rosso)

◆ initRayPos()

```
void initRayPos ( Mat *  normImg,
                  Mat *  rayPos
                )
```

viene segnata la posizione di tutti i pixel facenti parte dei 46 raggi precedentemente disegnati sull'immagine normalizzata

Parameters

normImg puntatore all'immagine normalizzata

rayPos puntatore alla matrice in cui inserire le posizioni utilizzate dai punti sui 46 raggi

◆ localMinima()

```
vector<double> localMinima ( vector< double >  vec )
```

filtro i valori che rappresentano dei minimi locali, in cui presi tre valori ho vec[i-1] > vec[i] < vec[i+1]

Parameters

vec vettore contenente i valori ottenuti dalla convoluzione dei raggi con il kernel gaussiano

Returns

vettore contenente gli indici dei valori eliminati da vec

◆ lowerEyelidDetection()

```
Mat lowerEyelidDetection ( Mat *  normImg )
```

tramite l'utilizzo della mean e standard deviation viene calcolato un valore di threshold. Tale valore viene usato per individuare la palpebra inferiore, operazione che viene effettuata solo se la standard deviation è > della mean deviation /4

Parameters

normImg spettro rosso dell'iride normalizzata

Returns

lowerEyelidMask è la maschera di individuazione della palpebra inferiore

◆ pixelConvolution()

```
double pixelConvolution ( Mat *  normImg,
                          int    x,
                          int    y,
                          int    ray
                        )
```

convoluzione con il kernel differenziale del pixel (x,y) del raggio numero ray disegnato precedentemente sull'immagine normalizzata normImg

Parameters

normImg iride normalizzata

x coordinata x del punto preso in considerazione sul raggio disegnato numero ray

y coordinata y del punto preso in considerazione sul raggio disegnato numero ray

ray numero del raggio disegnato (contati da destra verso sinistra)

Returns

valore risultante dalla convoluzione del pixel (x,y) del raggio numero ray con il kernel differenziale, la formula è: sqrt(pow(horizontalVal, 2) + pow(verticalVal, 2)). Si può anche fare un'approssimazione di questa formula, usando la seguente: abs(horizontalVal) + abs(verticalVal)

◆ removeOutliers()

```
void removeOutliers ( vector< int > *  vec_x,
                      vector< int > *  vec_y
                    )
```

elimino gli outliers (pixel i) in cui, presi tre pixel i-1, i, i+1 la posizione del pixel i ha la coordinata x[i-1] > x[i] < x[i+1] oppure x[i-1] < x[i] > x[i+1], ovvero elimino tutti i pixel che hanno un incremento o decremento improvvisi sull'asse delle x. Successivamente elimino anche i pixel in cui y[i-1] > y[i] < y[i+1] oppure y[i-1] < y[i] > y[i+1], ovvero che hanno improvvisi incrementi o decrementi sull'asse delle y

Parameters

vec_x vettore contenente in posizione i il pixel i-esimo (quindi la posizione contiene un pixel in particolare, questo è necessario perché in vec_y è presente la rispettiva coordinata y)

vec_y vettore contenente in posizione i il pixel i-esimo

◆ threshReflectionDetection()

```
Mat threshReflectionDetection ( Mat *  normImg,
                                int    ksize,
                                double c
                              )
```

tramite adaptive thresholding vengono individuati i riflessi all'interno dell'immagine normalizzata (spettro blu)

Parameters

normImg spettro blu dell'iride normalizzata

ksize grandezza della finestra per l'applicazione dell'adaptive threshold

c coefficiente sommato nel calcolo del threshold nella funzione OpenCV di adaptive thresholding

Returns

reflectionMask è la maschera dei riflessi individuati

◆ upperEyelidDetection()

```
Mat upperEyelidDetection ( Mat *  normImg,
                           string path
                         )
```

viene localizzata e individuata la palpebra superiore. Viene fatto disegnando sullo spettro rosso dell'immagine normalizzata dei raggi al quale verrà applicato, tramite convoluzione, un kernel differenziale (es: prewitt). I valori risultanti da tale convoluzione vengono valutati e per ogni raggio viene preso il valore massimo. Viene effettuata una scremetura di possibili outliers (questa fase sarebbe da migliorare) e viene applicata ai valori restanti la regressione polinomiale (terzo ordine). Una volta ottenuto il polinomio di terzo ordine, vengono calcolati i valori per la curva della palpebra superiore. Se non è possibile disegnare tale curva, si procede con la funzione ellipseFitting, in cui vengono presi anche i maxima speculari (y*-1) e viene disegnata un ellisse. Alla fine di tutto viene creata la maschera binaria di output e viene swappata per garantire una corretta posizione della palpebra inferiore e superiore per effettuare un merge di tutte le maschere.

Parameters

normImg iride normalizzata

path stringa contenente il path all'interno del quale verrà salvato il file contenente l'immagine della maschera di output

Returns

reversedUpperMask conterrà la maschera di output per l'individuazione della palpebra superiore, al quale però viene effettuata un'operazione di "swap" dei due settori della maschera, in modo tale da avere al centro superiore la posizione della palpebra inferiore (se presente) e negli angoli in alto la palpebra superiore individuata con la presente funzione.